IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s)	: Chandan Mathur	Confirmation No.	: 3240
Application No.	: 10/684,053	Group Art Unit	: 2183
Filing Date	: 2003-10-09	Examiner	: David J Huisman
Docket No.	: 1934-012-03	Customer No.	: 00996
Title	: COMPUTING MACHINE HAVING IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD		

DECLARATION UNDER 37 C.F.R. § 1.132

Introduction

[1]. This declaration is presented in four sections:

- Background
- Analysis of the Dretzka reference (U.S. 4,703,475)
- Analysis of the Briton reference (U.S. 6,216,191)
- Conclusions

[2]. Both of the analysis sections first summarize the teachings of the respective reference, and then explain how the respective reference is deficient relative to the pertinent claims of the patent application.

Standards Background

International Standards Organization's Open System Interconnection Specification

[3]. The International Standards Organization's (ISO) Open System Interconnection (OSI) Specification is relevant to the teachings of Dretzka. Dretzka's disclosure references, and is in the terms of, the OSI Specification. And even though not expressly stated in the patent application, the disclosed embodiments are consistent with the OSI Specification. Therefore, embodiments disclosed in the patent application and Dretzka may be explained and understood in the context of the OSI standard, which "provides

a description of the activities necessary for systems to interwork using communication media."[1]

[4]. The latest version of the OSI standard is available from the ISO for $157; please go to http://www.iso.org for more information. The specification identifier and title of the latest version of the ISO OSI standard is provided below.

ISO/IEC 7498-1:1994 Information technology -- Open Systems Interconnection -- Basic Reference Model: The Basic Model Edition: 1 | Stage: 90.93 | JTC 1 ICS: 35.100.01 Document available as of: 1994-11-17

OSI Seven-Layer Model –

[5]. This abstract model of networking (the Basic Reference Model) was formulated during the 1980s by the ISO. The OSI has had a major influence on developing the Internet protocols, but more recently it is the international standard associated with Open Architecture (OA) practices and the modern trend to deploy a system by building it from Commercial Off-The Shelf (COTS) hardware. These practices emphasize the design of portable software so that applications are independent from the underlying hardware and their interconnection networks. The practice of open architecture is often cited as the most affordable way to produce modern computer based systems. For example, the U.S. Navy has committed itself to acquisition reform through purchasing more affordable computer technologies and asserting OA practices and deployment of COTS hardware on navy ships. Rear Adm. Terry Benedict, the Program Executive Officer who was asked to lead this reform effort from the department of the Program Executive Office - Integrated Warfare Systems (PEO-IWS), identifies open architecture as the most affordable way to produce computer-based systems for military applications in this article from Defense Daily dated 5/19/2010, quoted word for word in the following paragraph.

[6]. *"LAYING THE GROUNDWORK FOR MAKING SURFACE SHIP COMBAT SYSTEMS OPEN: In the almost three years Rear Adm. Terry Benedict headed up the program executive office for integrated warfare systems he and his staff have been working to lay the foundation for the Navy's dive into opening up surface ship combat systems. Benedict has left to take command of Strategic Systems Programs, but before departing PEO IWS he told Defense Daily the Navy and industry are coming together on open architecture. "First and foremost I would consider the biggest accomplishment*

[1] ISO/IEC 7498-4

the partnership I have witnessed develop between government and industry," he said. "The Navy can have a vision but the Navy doesn't build combat systems, industry does. We set requirements [and] industry builds [to them]. What I have seen industry do is embrace, over time, the Navy's philosophy of moving toward open architecture combat systems of the future." But it hasn't been an easy path to trek. Industry, at times, has questioned the speed at which the Navy is moving toward incorporating open architecture. But Benedict believes the service has shown its dedication to the effort. "I think we had to prove that we were serious about this." Had the move toward open architecture been for the sake of open architecture as opposed to a very disciplined analytical systems engineering structure, it wouldn't have been successful, Benedict said. "I think we have moved in that direction and that has been with the tremendous assistance of industry. Their assistance has been, one, in support of execution on systems like Aegis, SSDS and LCS," he said. "Also they have helped us by pushing us a little bit with our leadership to ensure we are staying consistent with opening contracts up to competition." The Navy has also made a strong commitment to Congress, Benedict added. Every quarter the service files its report on the status of open architecture efforts across all the domains and enterprises, from space to littoral and mine warfare to the Marine Corps. "We made a strong commitment to Congress in our reports that we would open up systems for competition, that we would where possible drive in small businesses, and [look for] other opportunities ... for lower cost and technical, innovative approaches," he said. "I think, again, we are demonstrating that." There are challenges ahead for Capt. Jim Syring, who has been nominated to the rank of rear admiral and to take over the helm at PEO IWS. For example, shifting out of MILSPEC hardware and software presented challenges, Benedict said. Any system is made up of five pieces: Hardware, software, training, documentation, and people, Benedict said. "As we have moved out of MILSPEC and into COTS hardware, it requires that all five pieces adjust," he said. "The hardware is pretty easy, everybody understands the hardware. Instead of building a processor, you go buy a processor. Instead of writing your own software, you go use commercial application programs and integrate them. I think people get that." Where people have a harder time understanding the challenges of OA are in the training, documentation and people, Benedict added." (Geoff Fein, Defense Daily – 5/19/2010)

[7]. A significant part of the OSI standard is the abstract model itself, documented in OSI 7498 and addenda. The OSI model teaches using a process of sub-dividing a system into smaller standard parts. OSI calls these smaller standard parts "layers." Inside a single layer, the OSI standard calls for the existence of a collection of collaborating functions that work together to implement a "protocol." Each layer has only two interfaces: one interface to

the layer above it, and the other interface to the layer below it[2]. The summarized definitions given below are cited from the web page "The 7 Layers of the OSI Model", Webopedia, which can be found at http://www.webopedia.com/quick_ref/osi_layers.asp. A summary table is given below that comes from Wikipedia, which can be found at http://en.wikipedia.org/wiki/OSI_model.

1. Physical - Layer 1

[8]. This layer conveys the bit stream - electrical impulse, light or radio signal -- through the network at the electrical and mechanical level. It provides the hardware means of sending and receiving data on a carrier, including defining cables, cards and physical aspects. Fast Ethernet, RS232, and Asynchronous Transfer Mode (ATM) are examples of three different standardized protocols for physical-layer components.

2. Data Link - Layer 2

[9]. At this layer, data packets are encoded and decoded into bits. Layer 2 furnishes transmission protocol knowledge and management, and handles errors in the physical layer, flow control, and frame synchronization. The data-link layer is divided into two sub layers: The Media Access Control (MAC) sub-layer and the Logical Link Control (LLC) sub-layer. The MAC sub layer controls how a computer on the network gains access to the data and to permit its transmission. The LLC layer controls frame synchronization, flow control, and error checking.

3. Network - Layer 3

[10]. This layer provides switching and routing technologies, creating logical paths, known as virtual circuits, for transmitting data from node to node. Routing and forwarding are functions of this layer, as well as addressing, internetworking, error handling, congestion control and packet sequencing.

4. Transport - Layer 4

[11]. This layer provides transparent transfer of data between end systems, or hosts, and is responsible for end-to-end error recovery and flow control. It ensures complete data transfer.

5. Session - Layer 5

[12]. This layer establishes, manages and terminates connections between applications. The session layer sets up, coordinates, and terminates

---

[2] Of course, layers 1 and 7 are different since they are at the ends of the stack. A Layer 1 transmitter of a first unit always has an interface to a second unit's layer 1 receiver of a second unit, possibly through, *e.g.,* "transparent" network equipment, a wired channel, or a wireless channel. Layer 7 of a stack has only one interface to the layer 6 service below it.

conversations, exchanges, and dialogues between the applications at each end. It deals with session and connection coordination.

6. Presentation - Layer 6

[13]. This layer provides independence from differences in data representation (*e.g.*, encryption) by translating from application to network format, and vice versa. The presentation layer works to transform network data into the form that the application layer can accept, and vice-versa. This layer formats and encrypts data to be sent across a network, providing freedom from compatibility problems. It is sometimes called the syntax layer.

7. Application - Layer 7

[14]. This layer supports application and end-user processes. Communication partners are identified, quality of service is identified, user authentication and privacy are considered, and any constraints on data syntax are identified. Everything at this layer is application-specific. This layer provides application services for file transfers, e-mail, and other network software services. For example, Telnet and FTP are applications that exist entirely in the application layer. Tiered application architectures are part of this layer.

OSI Model –Summary Table

	Data unit	Layer	Function
Host layers	Data	7. Application	Network process to application
		6. Presentation	Data representation, encryption and decryption
		5. Session	Interhost communication
	Segment	4. Transport	End-to-end connections and reliability, Flow control
Media layers	Packet	3. Network	Path determination and ***logical addressing***
	Frame	2. Data Link	Physical addressing
	Bit	1. Physical	Media, signal and binary transmission

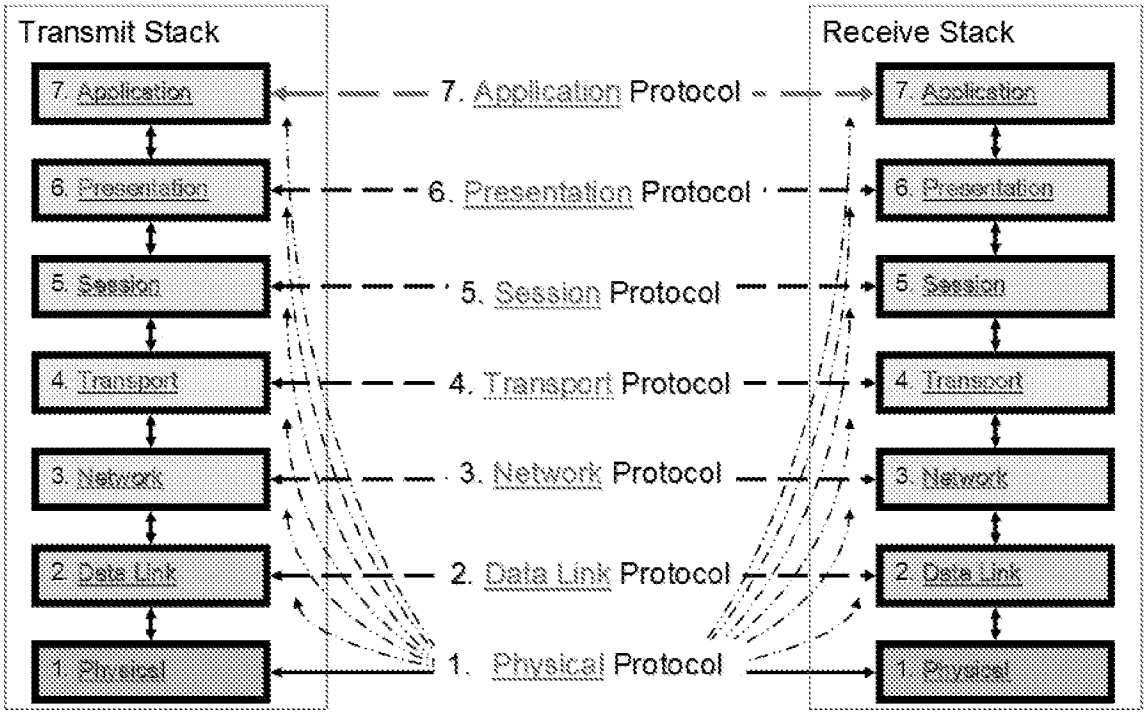
Functions, Protocols, Service Elements and the Stack of Services

[15]. A collection of ***functions*** will make up a ***service element*** that implements the ***protocol*** for one layer. The collection of all the adjacent ***service elements*** (one for each layer) is called the ***stack***. The elements of layer operation are specified in the OSI model; a service element is selected or designed to provide a specific protocol version where this version is identifiable.[3] The stack of service elements is also selected or designed to correctly interoperate; in other words the adjacent layers correctly interface together.

[16]. For example TCP/IP (an acronym for Transaction Control Protocol / Internet Protocol) is a stack of service elements that would be used by a software application to do transaction messaging over the internet. A software application can use many stacks of services; for example consider a computer with two network connections; one is a 100 Megabit Ethernet, while the other is the newer 40 Gigabit Ethernet. The first TCP/IP stack would be used for transaction messaging over a 100 Megabit Ethernet Network while a similar but uniquely different TCP/IP stack would be used for 40 Gigabit Ethernet transaction messaging. Yet the application software would interface at the top of both these different stacks using the exact same interface specification. This example illustrates an application having "*platform independence*." That is, the application software does not need to be modified when newer protocols, networks, or adapters are added to the computer system at the lower levels.

[17]. The figure below shows how applications can maintain a *persistent* application protocol over many generations of changes to the underlying network technology by isolating discrete functional changes to the layers where the protocol is changed. There are always two service-layer implementations of the protocol that must match: one is in the first unit's transmit stack, and the other is in the second unit's receive stack. The only physical linkage between the stacks is the layer 1 protocol; this is a real physical interface between the stacks. The other service-layer protocols (2, 3, 4, 5, 6, and 7) are only virtual interfaces between the stacks; their linkage to the only real interface is through the one interface with the layer immediately below. Likewise, the only way to interface to the application is through the one interface with the layer immediately above.

[3] ISO/IEC 7498-1: 1994(E) section 5.8 which starts on page 16.



[18]. From the given example adding the faster network (40 Gigabit Ethernet), layers 4, 5, 6 and 7 are unchanged. Also the interface between layers 3 and 4 is unchanged. However for 40 Gigabit Ethernet, the lowest layer (the physical means of transmitting bits) is radically different; therefore the physical layers (Layer 1) of both stacks have changed to a new common interface, and the functions inside the physical layers are also changed. Since the 40 Gigabit Ethernet standard specifies new Data Link Protocols, these would be implemented by new functions in Layer 2. Since data transmitted at this higher speed is more susceptible to transmission errors, the standard specifies new data parity codes that are used to correctly encode and recover the data. These new data parity codes are implemented by new functions included in Layer 3. The interfaces between Layers 1 and 2, and also between Layers 2 and 3 are now different for 40 Gigabit Ethernet when compared to the 100 Megabit Ethernet interfaces between these layers.

[19]. A function is distinctly different than a layer's service element. The OSI defines three different kinds of data types for the functionality that should be included in a layer: *Protocol-control-information* (PCI); *Protocol-data-unit* (PDU); and a *Service-data-unit* (SDU).[4] Consider a *Network* service element for *Layer 3*: it contains many functions, including:

♦ Mapping logical destinations (*i.e.*, transmitting data from node to node) to create the logical paths for this transmission.

♦ Mapping these logical paths onto virtual circuits that will activate later in layers 1 and 2.

[4] ISO/IEC 7498-1 : 1994(E) section 5.6 page 15.

- Providing the network addressing for the transmission, used in layers 1 and 2.
- Providing data error handling, (*e.g.*, encoding parity bits, or detecting / correcting parity bit errors).
- Providing the transmission service related to error handling.
- Providing persistent registration for configuration data.
- Providing for congestion control.
- Providing packet sequencing.
- Providing the interface to Layer 4.
- Providing the interface to Layer 2.

[20]. These functions inside a service element are selected in accordance with a specified protocol for that layer of service. These functions inside the service element are designed to collaborate. For example the data-parity encoding function of the transmit stack matches the data-parity decoding function of the receive stack. If a non-correctable parity error is detected, then functions are invoked for both the data-error handling of the receive stack, and the transmission service related to error handling that communicates the error back to the transmit stack. It should be readily apparent now that no single function is sufficient to perform a non-trivial protocol.

[21]. Some subset of functions from the following list of functions would be integral to a Layer 5 Service-Element:[5]

Connection Establishment & Release	Flow Control	Error Detect & Notification
Suspend	Expedited Segmenting	Reset
Resume	Blocking	Routing
Muxing & Splitting	Concatenation	Quality of Service
Normal Data Transfer during Establishment	Sequencing Acknowledgement	

[22]. Many people will fail to distinguish the scope of the composition of functions that make up the service element for one layer. A common mistake is made when a person discusses a network function as if it is the complete service element. A service element is really a composite of many collaborating functions that together implement a specified protocol.

[23]. One recurring difficulty is protocol mismatching that occurs when one of the stacks is implemented in hardware, but the other stack is in software: hardware – software integration is historically more difficult than software -

[5] ISO/IEC 7498-1: 1994(E) section 5.8.6 which starts on page 20.

software integration. Hardware – software integration often requires much more testing, and a more sophisticated test bench to validate that a protocol is correctly matching in two different domains of engineering. One difficulty that often occurs for the hardware engineer is when the hardware functions need to set the data types for the service-element; the *Protocol-control-information* (PCI); *Protocol-data-unit* (PDU); and a *Service-data-unit* (SDU). The difficulty derives from the fact that data is typically stored on a disk drive, and this data is normally accessed by software routines and the operating system to register the data. This means the hardware designer must often rely on software services to read the data and transfer the data to the hardware where it would then be registered. Since the service-entity data isn't registered yet, the service entities must correctly transfer the service data types without this critical data being there. This fundamental gap often must be addressed to solve the protocol mismatching inherent in heterogeneous hardware-software stacks.

The Historical Difficulties Practicing the OSI Standard

[24]. In spite of the enduring use of the OSI Standard, there have been too many cases where either the hardware or the software violated one of the OSI rules. The most common consequence occurs rapidly; within a few months or perhaps a year, these OSI violating products may become obsolete because they cannot be easily updated with new features. The consumer of the product is left with either costly upgrades, or archaic performance or features. Soon these products fail to sell because they are not competitive. These commercial failings have had technical consequences. For example, many engineering teams now conduct peer review during product design to ensure that the OSI standard and/or other standards are practiced correctly.

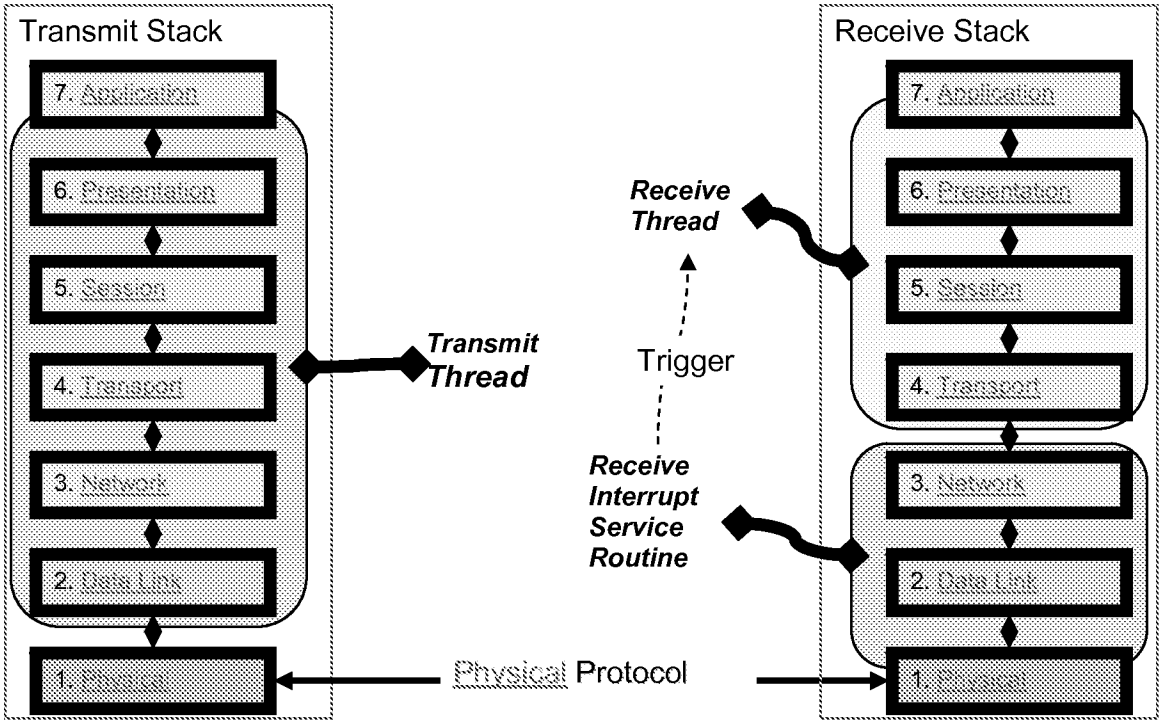
[25]. The Dretzka and Briton references originate from a time when engineering peer reviews were not routinely conducted, at least not to confirm compatibility with the OSI 7-layer model. As discussed in more detail below, both of these references break at least some of the OSI rules. In the time when these references were written, these types of mistakes were more easily overlooked. Today, mistakes like these are typically considered unacceptable in a computer-technology practice.

[26]. Here are some common mistakes regarding the OSI Model:

- A function that is designed to be the complete service element
- A service element that requires configuration data that can't be set (*i.e.*, a service element that calls for configurability is unconfigurable).
- Interfacing a layer with a non-adjacent service layer (*e.g.*, interfacing Layer 1 to Layer 4, thus skipping Layers 2 and 3)
    - Often, this non-adjacent layer is called the "application" even though this layer may not be the Application Layer 7. Per OSI, only Layer 6 interfaces with the Application Layer 7

- Mismatching the protocol between transmit and receive stacks
- Broken Threading of the Stack

[27]. This last issue - Broken Threading of the Stack requires a bit more discussion. This is a more recent issue that comes with the highly networked computer systems. Engineers and software developers are now taught away from breaking up the stack into multiple threads. For best network performance, it is critical to keep consistent and timely execution the service elements across each of the stacks (*i.e.*, keeping the service elements together in a single thread of execution). Doing so avoids transmission problems that inevitably occur when a highly multitasking processor switches away from the stack execution and introduces unnecessary delays between executing the service elements. A single thread of execution helps to minimize the message buffering space required, and simplifies the multitask scheduling of the software system. The security of the Receiving Stack is improved because there is less opportunity for buffer over-runs. The figure below illustrates correct single threading for the transmit stack and the receive stack.



[28]. Typically today, hardware provides Layer 1 through Layer 3, whereas historically it was Layer 1 and parts of Layer 2. Software provides the remaining layers. For the Transmit Stack, at least from the edge of the stack in Layer 7 and all the layers below it (down to the hardware) are combined into a single thread of execution – greatly increasing the rate of data transfer out of the unit. For the Receive Stack, interrupts are generated by the

network adapters that need to service the discrete network transmission. The packets arrive quickly, they arrive out of order, and they arrive without timing constraints. This means that Layers 3 and below need to be bundled together into an Interrupt Service Routine (ISR) in order to most rapidly and completely receive each and every transmitted packet. With the first received packet, the ISR triggers the execution of a single thread containing all the remaining service layers from Layer 4 up to at least the edge of the stack in Layer 7. Doing this greatly increases the rate of data transfer into the unit, and greatly reduces the latency of processing network transmissions. By using the single Receive Thread, there is only one interrupt in the system that needs to be serviced – the one for the network adapter that triggers the Receive Interrupt Service Routine for the next packet. In this way the Receive Interrupt Service Routine and the Receive Thread are a simple and tightly coupled executing package.

Analysis of the Dretzka reference

Dretzka's Teachings

[29]. One of Dretzka's teachings is "a multilink interprocessor communication protocol that allows a pair of processors to increase the speed of communications by using multiple physical links in parallel."[6] Dretzka further specifies that these multiple physical links operate completely independent of one another[7], and that the protocols must include switched network transmission[8], data packetizing[9], and packet sequencing.[10] Dretzka invokes the OSI model as the base communication standards reference in his specification.[11]

[30]. Dretzka teaches that the interface circuits are external circuitry that normally are interfaced by adapters attached to the processor, but are not interfaced directly the processor.

Overview

[31]. Dretzka teaches speeding up data transmission by spreading the data across multiple independent serial transmission links. Dretzka references prior art that teaches about data networks where data packets are routed to

---

[6] Quoted from Patent 4,703,475 – Abstract.
[7] Multiple Links that operate independently: references appear in Col 2 lines 27-32, lines 35 – 39, Col 3 lines 7 – 11, lines 28 - 31.
[8] Protocols must include switched network transmission: references appear in Col 2 lines 40 – 45, Col 4 lines 18-36.
[9] Protocols must include data packetizing: references appear in Col 2 lines 19-26, lines 46-65; Col 3 lines 31-39.
[10] Protocols must include packet sequencing: references appear in Col 2 lines 46-65; Col 3 lines 11 - 20, lines 31-39.
[11] OSI references appear in Col 1 lines 20-37, Col 3 line 50 through Col 4 line 36, and Figures 3, 4, 5, 6, and 8 that indicate which layers of the OSI model are affected in the drawing.

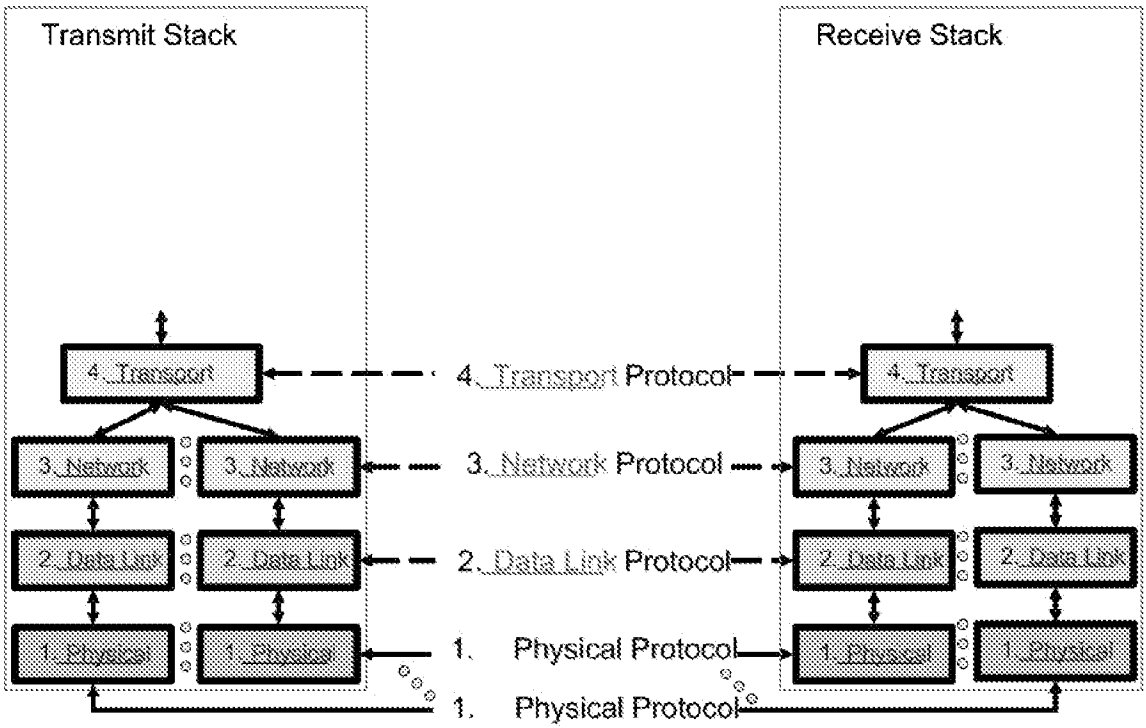
different destinations on the network using network addresses. Dretzka builds upon this prior art; first by giving each node on the network multiple independent serial transmission links, and then teaching a method of taking a message, and then sequencing the derived packets across the multiple serial transmission links.

Multiple Serial Bus Interface

[32]. Dretzka teaches using a modified OSI stack, as shown below[12], where there are multiple implementations of service elements for layers 1, 2, and 3 corresponding to the number of serial transmission links on the nodes. The function for distributing the packets across the multiple serial transmission links is in Dretzka's transmit stack in layer 4.[13] The matching function for reassembling the transmitted message received across the serial transmission links is in Dretzka's receive stack in layer 4.[14] Dretzka limits the specification to the smallest subset of functions in layers 2, 3 and 4 that are needed to speed up data transmission by spreading the data across multiple independent serial transmission links.

[33]. Recent developments in the network and computer industries now favor many aspects of the Multiple Serial Bus approach taught by Dretzka. This approach is now used in the Fast Ethernet standard, the Serial Rapid I/O standard, the PCI Express Standards, and many Serial Disk Interface Standards. Each of these standards utilizes different kinds of network configurations and transaction techniques suited to best serve the purposes of these busses and networks. Dretzka's teaching has a limited focus on the application of the Multiple Serial Bus approach. Dretzka's approach depends on many other functions in order to operate correctly. Any complete Network or Buss specification could not rely solely on Dretzka's approach; there are so many functions needed that Dretzka did not anticipate at the time of his teaching, and, therefore, that Dretzka does not discuss or even consider.

[12] This figure is derived from the OSI Model, and from Dretzka's figures 3 and 4.
[13] Derived from Dretzka's figure 3.
[14] Derived from Dretzka's figure 4.



[34]. Dretzka is inconsistent in specifying the interface above the Layer 4 (Transport Service Element). At first in Col 4 lines 6 - 16, he specifies that the Layer 5 (Session Service Element) is interfaced to his Layer 4 functions. This is in accordance with the OSI Model. But then in his figures 1, 2, 3, 4, 5, and 6 and the associated descriptions he indicates that his Layer 4 functions are interfaced to the application (Layer 7), which violates the OSI Model. These errors in the Dretzka figures are addressed individually in the paragraphs that follow below.

[35]. If we further examine Dretzka's specification against the list of functions that the OSI Model says should be present, we can see that Dretzka is properly specifying some functions, but not all of the functions. For brevity, let's examine Dretzka's specification against the list of functions that should be present in the Layer 3 service-element. Dretzka properly specifies only two of the Layer 3 functions. In fact, Dretzka is silent as to most of the functions of the Layer 3. But he does not discuss how to interface these two functions to the others that should be in Layer 3. Table 1 below lists in the right column the Layer 3 functions specified by OSI standard, and lists in the left column the inclusion or omission of these functions, as discussed by Dretzka. As one can see, Dretzka is silent as to a majority of these OSI specified Layer 3 functions. Specifically, Dretzka provides no teaching as to how, or even if, his device implements these omitted Layer 3 functions or their interfaces.

Table 1 Network Service-Element - Layer 3 Functions	
Dretzka's Spec.	**OSI Standard Functions for Layer 3**

Table 1 Network Service-Element - Layer 3 Functions	
Dretzka's Spec.	**OSI Standard Functions for Layer 3**
Included	Mapping logical paths onto virtual circuits.
Included	Packet sequencing.
OMITTED	Mapping logical destinations (i.e., transmitting data from node to node) to create the logical paths for this transmission.
OMITTED	Network addressing for the transmission, used in layers 1 and 2.
OMITTED	Data error handling, (e.g., encoding parity bits, or detecting / correcting parity bit errors)
OMITTED	Transmission service related error handling.
OMITTED	Persistent registration for configuration data.
OMITTED	Congestion control.
OMITTED	The interface to Layer 4.
OMITTED	The interface to Layer 2.

Dretzka Specifies a Network Link from one CPU to a second CPU

[36]. Dretzka starts his abstract with the idea that there is: "a multilink interprocessor communication protocol that allows a pair of processors to increase the speed of communications by using multiple physical links in parallel."[15] The illustration of the three remote switching modules 10, 20, and 30 in Dretzka's Figure 1 confirms that Dretzka's network links occur from a first node (*e.g.*, module 10) to a second node (*e.g.*, module 20) or even a third node (*e.g.*, module 30).

Figure 1

[37]. In Dretzka's description of this figure, he describes that no host switching is illustrated in this figure[16], and that the links could be part of a larger network by using prior art. Dretzka describes that the links illustrate a two-way transmission[17]. Dretzka stipulates that these are "paired links" and that there would be multiple paired links between "remote switching modules"[18] that are commonly called a ***Node*** today. There are three distinct nodes illustrated (identified as 10, 20 and 30). Dretzka discusses applications and OSI services that are running on processors (identified as 11 and 12) that are contained inside these nodes, using "Inter-Processor Communication."[19] There is also an external interface of the node to the facility network, which Dretzka calls a DFI (Digital Facility Interface); these are identified on the

[15] Quoted from Patent 4,703,475 – Abstract.
[16] Col 5, lines 20-27.
[17] Col 5 lines 27-33.
[18] Col 5 lines 33-37.
[19] Col 6 lines 9-15.

drawing as 14-0, 14-4,14-5, 14-m, 24-0, 24-4,24-5, 24-n. A pair of DFI, specifically one in each node, provides a singular paired link between two of the nodes (10, 20, or 30). Today, a modern network adapter would include multiples ones of Dretzka's DFIs.

[38]. The three nodes (units 10, 20, and 30) are illustrated as being identical. All three have exactly the same resources and topologies, and the same number and quality of interfaces. One must conclude that as soon as there are more than two nodes in communication, there must be functions included in the layer protocols to provide unique addresses enabling the data transfer to one destination node and not to the other possible destination nodes. As discussed previously, these multi-node addressing and routing protocol functions occur across multiple layers. Dretzka does not sufficiently discuss the dependencies of his Multiple Serial Bus approach upon the multi-node addressing and routing functions in his specification. Therefore, although one of skill in the art would know that Dretzka's data packets must include a node/network address even though Dretzka does not specifically mention such an address, he would be uncertain about Dretzka's dependence on the source and formatting of the network address, and Dretzka's handling or transformation of the necessary network address or routing information required by his Multiple Serial Bus approach. Also Dretzka does not teach methods for creating or translating information to derive the necessary address or routing destinations. Since he does not teach that there is a critical information interface, or an alternative method of deriving the information or other critical functions, the specification here is incomplete.

Figure 2

[39]. This figure is a simplified version of Figure 1. The third node (unit 30) is not illustrated. The OSI service-layers 2 – 7 are again not illustrated. Operating System services are again not illustrated. Dretzka indicates OSI service-layers 2 – 7 are running in the processors (11 and 21)[20].

[40]. The next step for one ordinarily skilled in the art is to identify the service-layer data types (*i.e.*, the PCI, PDU and SDU discussed earlier); this data that needs to associate with Dretzka's functions and the other functions that he depends on. These data types need declarations that will allocate storage somewhere in memory (12 & 13). Then one ordinarily skilled in the art needs to identify the source of configuring the new data, which configuration data is typical stored on hard drives. It is a notable omission that Dretzka does not illustrate these data type dependencies or configuration data dependencies. In conclusion about this figure, Dretzka does not sufficiently discuss the dependencies of his Multiple Serial Bus approach upon the multi-node addressing and routing functions in his specification, as discussed for figure 1.

[20] Col 7, lines 6 -17.

Figure 3

[41]. Although this figure illustrates the transmit stack, Dretzka discusses only a subset of layers for this stack, and only a small subset of the functions that the OSI standard specifies for these layers. The OSI service layers 2 – 4 are illustrated - indicated in the left-most column by the ***Level #***. This figure also lists Dretzka's functions performed by the processor (11 and 21)[21]. However, Dretzka fails to acknowledge that his list of functions not a complete list for implementing the service-layer indicated in the left column. Dretzka is silent to the fact that there must be other functions that work alongside or that support his functions in each layer.

[42]. Dretzka does not acknowledge how the many other functions that must be in the service-layer should collaborate with his functions, and he does not list any critical dependencies with his functions, neither within the service-layer nor across the service-layer boundaries.

[43]. For example, missing functions from layer 4 include the end-to-end connections, reliability and flow-control features. Missing functions from layer 3 include the path-determination and logical-addressing features. Missing functions from layer 2 include physical-addressing features and a physical-address map. The Layer 1 dependencies are not indicated, even though the physical cables of Layer 1 are illustrated (at the bottom of the figure: 40-0 and 40-4) and the physical parts of the DFI (14-0 through 14-4) are illustrated. Layer 1 is not indicated correctly in this drawing.

[44]. In Dretzka's figure 3, an interface is illustrated (using the vertical downward arrow) only between the Dretzka functions (written inside each box) that occur in the different layers (Level 4, Level 3, and Level 2.5). However, this is misleading because Dretzka fails to acknowledge that his illustrated interface (vertical downward arrow) is not the complete set of data for implementing the service-layer interface (Level 4 to Level 3, Level 3 to Level 2.5). Again, Dretzka does not sufficiently discuss the dependencies of his Multiple Serial Bus approach upon the multi-node addressing and routing functions in his specification, as was previously discussed for figure 1.

[45]. Dretzka also makes a most serious OSI model violation at the top of this figure. This transmit stack is declared to provide Inter-host Communications, but omits Layers 5 and 6 by directly interfacing Layer 4 to the Application Program (yet Layer 7 is omitted from the figure even though Layer 7 is the Application Program layer).

[46]. Dretzka fails to acknowledge that his functions depend on data for the *Protocol-control-information* (PCI), *Protocol-data-unit* (PDU), and *Service-data-unit* (SDU) data types. Dretzka also fails to recognize his dependence

[21] Col 7, lines 6 -17.

on an Operating System to retrieve the data he needs so that the PCI, PDU, and SDU data types for his functions can be configured.

[47]. Dretzka has been silent for most of the functionality that comes with OSI compliance, functionality that the reader must be careful about. One of skill in the art can not determine what the missing functions are, and cannot be certain what Dretzka is depending on.

Figure 4

[48]. This figure, similar to figure 3, illustrates an insufficient subset of the receive stack. Dretzka illustrates the matching behavior to the functions shown in Figure 3. The OSI service layers 2 – 4 are illustrated - indicated in the left most column by the ***Level #***. This figure also lists Dretzka's functions performed by the processor (11 and 21)[22]. However, Dretzka fails to acknowledge that his list of functions is not a complete list for implementing the service-layer indicated in the left column. Dretzka is silent as to the fact that there must be other functions that work alongside or that support his functions in each layer.

[49]. Dretzka does not acknowledge how the many other functions that must be in the service-layer should collaborate with his functions, and he does not list any critical dependencies with his functions, neither within the service-layer nor across the service-layer boundaries.

[50]. For example, missing functions from layer 4 include the end-to-end connections, reliability, and flow-control features. Missing functions from layer 3 include the path-determination and logical-addressing features. Missing functions from layer 2 include physical-addressing features and a physical-address map. The Layer 1 dependencies are not indicated, even though the physical cables of Layer 1 are illustrated (at the bottom of the figure: 40-0 and 40-4) and the parts physical of the DFI (14-0 through 14-4) are illustrated. Layer 1 is not indicated correctly in this drawing.

[51]. In Dretzka's figure, an interface is illustrated (using the vertical upward arrow) only between the Dretzka functions (written inside each box) that occur in the different layers (Level 4, Level 3, and Level 2.5). However, this is misleading because Dretzka fails to acknowledge that his illustrated interface (vertical upward arrow) is not the complete set of data for implementing the service-layer interface (Level 4 to Level 3, Level 3 to Level 2.5). Again, Dretzka does not sufficiently discuss the dependencies of his Multiple Serial Bus approach upon the multi-node addressing and routing functions in his specification, as was previously discussed for figure 1.

[22] Col 7, lines 6 -17.

[52]. Dretzka again makes a most serious OSI model violation at the top of this figure. This transmit stack is declared to provide Inter-host Communications, but omits Layers 5 and 6 by directly interfacing Layer 4 to the Application Program (yet Layer 7 is omitted from the figure even though Layer 7 is the Application Program layer).

[53]. Dretzka fails to acknowledge that his functions depend on data for the *Protocol-control-information* (PCI), *Protocol-data-unit* (PDU), and *Service-data-unit* (SDU) data types. Dretzka also fails to recognize his dependence on an Operating System to retrieve the data he needs so that the PCI, PDU, and SDU data types for his functions can be configured.

[54]. Dretzka has been silent for most of the functionality that comes with OSI compliance, functionality that the reader must be careful about. One of skill in the art can not determine what the missing functions are, and cannot be certain what Dretzka is depending on.

Figure 5

[55]. This figure illustrates the message registrations across an insufficient subset of the transmit stack. The OSI service-layers 2 – 4 are illustrated - indicated at the top of the figure by the ***Level #***. Again, Dretzka does not discuss the dependencies of his Multiple Serial Bus approach upon the multi-node addressing and routing functions in his specification, as was discussed for figure 1.

[56]. Again, Dretzka makes the same serious OSI model violation at the left of this figure by omitting Layers 5, 6 and 7.

Figure 6

[57]. This figure illustrates the message registrations across an insufficient subset of the receive stack. The OSI service-layers 2 – 4 are illustrated - indicated at the top of the figure by the ***Level #***. Again, Dretzka does not sufficiently discuss the dependencies of his Multiple Serial Bus approach upon the multi-node addressing and routing functions in his specification, as was discussed for figure 1.

[58]. Again, Dretzka makes the same serious OSI model violation at the right of this figure by omitting Layers 5, 6 and 7.

Figure 8

[59]. This figure illustrates how data received from the paired links (40-1 through 40-4) are transformed up through the receive stack registers through layer 3. In modern systems there are dozens to hundreds of these paired links. Dretzka fails to teach about a necessary mapping of the relevant physical links to a specific Stack of Services (the bundled logic). For example, one service entity in layer 4 is mapped to several service entities in layer 3.

This mapping must correspond to the physical pins of the paired links being serviced. Again, Dretzka does not sufficiently discuss the dependencies of his Multiple Serial Bus approach upon the multi-node addressing and routing functions in his specification, as was discussed for figure 1.

[60]. No other figures are analyzed.

Critical features omitted or taught incorrectly

Layer 4 Should Never Interface With The Application Layer

[61]. Dretzka again makes a most serious OSI model violation at the top of figure 3 and figure 4. His transmit and receive stacks are declared to provide Inter-host Communications, but Dretzka omits Layers 5 and 6 by directly interfacing Layer 4 to the Application Program (even though Layer 7 is the Application Program layer).

Composite Functionality in a Service-layer is not taught

[62]. Dretzka does not specify how network node identification (*i.e.*, transport-address, network address, etc.) of a message is provided to his functions, or how his functions utilize endpoint information to then map network addresses. These dependencies of collaborating functions are not identified inside the service-layers he uses. There are critical data types missing, including for layer 4 of the transmit stack', a mapping table that is used to transform the "End transport-address" that was received across the interface from layer 5 into the "End network-address" that is then sent across the interface to layer 3.[23]

[63]. Dretzka fails to acknowledge that his functions depend on data for the *Protocol-control-information* (PCI), *Protocol-data-unit* (PDU), and *Service-data-unit* (SDU) data types. Dretzka also fails to recognize his dependence on an Operating System to retrieve the data he needs so that the PCI, PDU, and SDU data types for his functions can be configured.

[64]. Dretzka has been silent for most of the functionality that comes with OSI compliance.

Complete Interfaces Between Service-layers Is Not Taught

[65]. For the Transmit Stack, Dretzka completely omits teaching about his dependence upon the receipt of the "End transport-address" from Layer 5 into Layer 4, and the transformation of the "End transport-address" into the "End network-address" which is then transferred between layers 4 and 3.[24] For the Receive Stack, Dretzka should be teaching the receipt of the "network-

[23] ISO/IEC 7498-1: 1994(E) section 7.4.4.2 Addressing which starts on page 39.
[24] ISO/IEC 7498-1: 1994(E) section 7.4.4.2 Addressing which starts on page 39.

address" from Layer 3 into Layer 4, and its transformation into the "transport-address" which is then transferred between layers 4 and 5.[25]

[66]. It is the "End transport-address" information passed from Layer 5 to Layer 4 that Dretzka depends on for his Multiple Serial Bus approach to do the multi-node addressing and routing functions; dependencies missing from his specification (as preciously discussed for figure 1). From this information, Node 10 can determine which set of Multiple Serial Busses to use for sending data to either Node 20 or Node 30, and the switched network can use the network addresses in the header to route the packets to either Node 20 or Node 30.

[67]. In Summary, Dretzka must have a node address, but he does not teach it; for switched networks Dretzka's data packets must obtain a header with a specific address (*i.e.,* Node 20 vs. Node 30 when the message comes from Node 10); this address must be derived somehow from the "End transport-address" data provided by Layer 5.

Operating System Dependencies Are Not Taught

[68]. There are critical operating system dependencies for Service Entity data types missing; such as the configuration file data for the transmit stack's layer 4 ***mapping table*** that is used to transform the "End transport-address" from layer 5 into the "End network-address" used in layer 3.[26]

Analysis of the Briton reference

Briton's Teachings

[69]. One of Briton's teachings is "a field programmable gate array (FPGA) that has an interface circuit that allows signals to be transmitted directly between the FPGA and the processor."[27] Briton further specifies that this interface circuit eliminates the external circuitry that normally would be in between the processor (102) and the FPGA (104)[28], and that this relationship to the processor (102) is singular and dedicated[29] - no other processor (102) shares the FPGA (104) resource. The Briton interface is further specified as a parallel type of memory circuit interface[30].

---

[25] ISO/IEC 7498-1: 1994(E) section 7.4.4.2 Addressing which starts on page 39.
[26] ISO/IEC 7498-1: 1994(E) section 7.4.4.2 Addressing which starts on page 39.
[27] Quoted from Patent 6216191 B1 – Abstract.
[28] Col 1, lines 28 – 37, Col 2 lines 11 - 12.
[29] Col 1, lines 30 - 31.
[30] Col 2 lines 9 -13. In Col 1 Lines 64 -66, Briton references to the PowerPC and Intel i960 processors which define this bus as a type of memory bus. This PI Bus is defined in *Table 5-9. Input/Output Signal Descriptions* starting on page 66 of the "IBM PowerPC 970MP RISC Microprocessor Datasheet," Version 1.3, dated January 17, 2008; available on line at https://www-

Overview

[70]. Briton teaches simplifying data transmission between one processor (102) and an FPGA (104) by interfacing them directly using a parallel buss[31]. Briton teaches directly translating the electrical buss signals for the user defined logic without using any data or address registers[32]. The data registers or buffers must therefore be part of the user defined logic. One of ordinary skill in the art would know the registers or buffers need to be included in his Microprocessor Interface (100), and that good practice dictates putting registers between components to isolate the designs. Yet Briton teaches against good practice; that the application logic on the FPGA (104) must interact with his internal busses without address registers, and data registers.Briton teaches the Microprocessor Interface (100) is built into the FPGA (104) and does not occupy user space in the FPGA (104)[33]. Therefore, the interface 104 is a most basic physical layer interface.

[71]. The Briton reference is very narrowly focused on non-standard custom interfaces between one processor (102) and an FPGA (104) [34]. Briton does not once refer to the OSI Model. By teaching that the application logic must interface with his busses and system registers (306), Briton does not comply with the OSI seven layer model. By specifying that the FPGA (104) must interface directly with the processor (102) [35], Briton excludes applying his approach to the open systems interfaces of the OSI Model (*e.g.*, Ethernet, PCI Express, or Rapid IO) and Briton excludes switched interfaces taught by Dretzka.

Parallel Bus Interface

[72]. Briton specifies using a parallel buss to directly couple the FPGA (104) to one processor (102). The Briton reference, although not compliant with the OSI model, teaches registering the buss signals in a way that would be equivalent to functions in Layer 2 (Data Link - Logical Link Control (LLC) sub-layer) of the OSI model. The distinction of this Layer 2 LLC equivalency is important since Dretzka relies upon the seven layer OSI Model. This equivalency allows for a context to consider mixing the teachings of Dretzka

---

01.ibm.com/chips/techlib/techlib.nsf/techdocs/B9C08F2F7CF5709587256F8C006727F1/$file/970MP_DS_DD1.1x_v1.3_17Jan2008_pub.pdf.

[31] Col 2 lines 9 -13. In Col 1 Lines 64 -66, Briton references to the PowerPC and Intel i960 processors which define this bus as a type of memory bus. This PI Bus is defined in *Table 5-9. Input/Output Signal Descriptions* starting on page 66 of the "IBM PowerPC 970MP RISC Microprocessor Datasheet," Version 1.3, dated January 17, 2008; available on line at https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/B9C08F2F7CF5709587256F8C006727F1/$file/970MP_DS_DD1.1x_v1.3_17Jan2008_pub.pdf.

[32] Not shown on Figure 2 or Figure 3.

[33] Col 1, lines 35 – 40.

[34] Col 1, lines 28 – 37, Col 2 lines 11 - 12.

[35] Col 1, lines 28 – 37, Col 2 lines 11 - 12.

and Briton. Briton also makes use of the existing art for implementing equivalent Layer 1 physical interfaces for his parallel bus electrical signals.

Specifically a Memory Bus from a CPU

[73]. This OSI Model equivalency is only partial, because Briton specifies that the application logic must use the address and data registers (306) of a memory bus. A memory bus is tightly controlled by its processor (102) to read and write to memory addresses, so the OSI concept of independent transmit and receive stacks in units located on both sides of the bus do not fit very well into Briton's device. Nonetheless, there is a very loose equivalency.

[74]. Briton teaches registering the FPGA (100) buss signals received at the Microprocessor interface (100), which is one of the roles described for the OSI model Layer 2 functionality for Logical Link Control (LLC) sub-layer in the receiver stack. In the OSI model, the LLC layer controls synchronization, flow control, and error checking, however Briton does not teach about error checking. Also missing from Briton's teaching about the FPGA (104) is the MAC sub layer of the OSI model, which would control how the device on the buss gains access to the data and permission to transmit it. Briton imposes this "data access" responsibility (of an OSI MAC) upon the application logic, which would be a violation of the OSI Model, since the application logic (layer 7 by definition) is interfaced to a sub-layer of layer 2.

[75]. More significantly, the permission to transmit can never be given to the FPGA (104), because the processor (102) has master control of the memory buss. For the FPGA (104) to transmit a block of data to the processor (102), the processor (102) must first set registers (306), in the Microprocessor interface (100) to initialize the transfer, and then grant the Microprocessor interface (100) temporary control of the buss to perform the data block transfer. The FPGA (104) does not have a transmit stack, since it can only transmit the data selected by the processor (102), and can only transmit when the processor (102) grants the buss.

[76]. Briton's teachings focus primarily on the FPGA (104). Britan provides no detail on the services, or functions that must be present in the processor (102) to make use of his circuit.

[77]. The addresses of Briton's teachings are for registers (306) associated to memory addresses, and user logic addresses; Briton's addresses are not for devices associated to a network.

[78]. The figure below identifies the equivalent OSI layers that Briton discusses, where the transmit stack is always in the processor (102) and the receive stack is always in the micro-processor interface (100) inside the FPGA (104). Briton's teachings are clearly not OSI compliant.

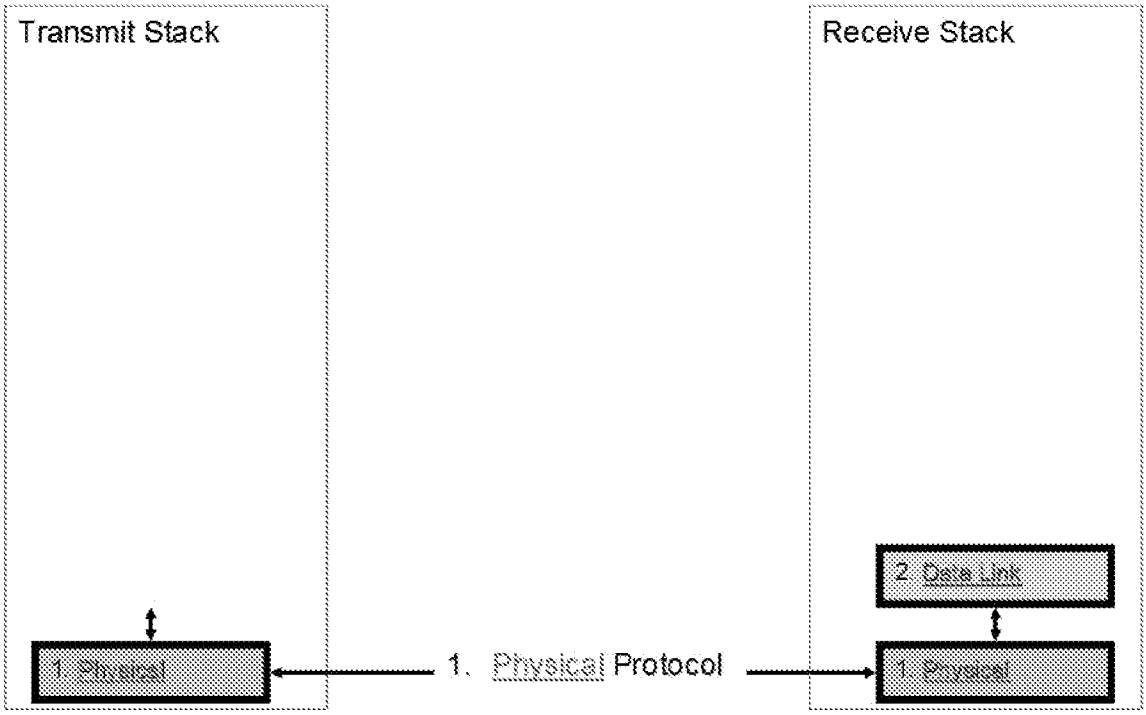


Figure 1

[79]. A processor (102) sends signals across a Processor Interface (PI) buss[36] that interfaces to an FPGA (104) through a Microprocessor Interface (100). Briton constrains the scope of his solution to attaching the FPGA (104) to the PI buss without any additional logic[37]. The novelty in this figure is merely the extremely minimal footprint - adding only one chip to the circuit board; the FPGA (104).

Figure 2

[80]. The Microprocessor Interface (100) block diagram solution is shown specifically for the Processor Interface (PI) buss for the Intel i960 Processor. Briton is teaching techniques that are very basic (*e.g.*, the bidirectional data buss[38] implementation (not marked), the bus timing and control logic (202), the address decode logic (204), and the system registers (206)); the novelty in this figure is the extremely minimal circuitry implementation that would need to go into the Microprocessor Interface (100). It is notable that the user

[36] Col 2 lines 9 -13. In Col 1 Lines 64 -66, Briton references to the PowerPC and Intel i960 processors which define this bus as a type of memory bus. This PI Bus is defined in *Table 5-9. Input/Output Signal Descriptions* starting on page 66 of the "IBM PowerPC 970MP RISC Microprocessor Datasheet," Version 1.3, dated January 17, 2008; available on line at https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/B9C08F2F7CF5709587256F8C006727F1/$file/970MP_DS_DD1.1x_v1.3_17Jan2008_pub.pdf.

[37] Col 2 lines 12-13.

[38] Col 5, line 21.

defined logic interface is not registered. Using registers would have been the standard (to one ordinarily skilled in the art) technique used to isolate different design segments.

Figure 3

[81]. The Microprocessor Interface (100) block diagram solution is shown specifically for the Processor Interface (PI) buss for the PowerPC Processor. Briton is teaching techniques that are very basic (*e.g.*, the bidirectional data buss[39] implementation (not marked), the bus timing and control logic (302), the address decode logic (304), and the system registers (306); the novelty in this figure is the extremely minimal circuitry implementation that would need to go into the Microprocessor Interface (100). It is notable that the user defined logic interface is not registered. Using registers would have been the standard (to one ordinarily skilled in the art) technique used to isolate different design segments.

Critical features omitted or taught incorrectly

[82]. Briton's FPGA (104) applications are tightly constrained to situations where the FPGA (104) is attached to a processor (102) through a memory buss that has separate address and data lines, and where error correction is not used, and where an address is provided with every data word.

[83]. Briton teaches about transactions[40] across the processor interface buss. Briton's transactions are not the network transactions of the OSI Model, but are instead pure memory transactions; where data stored in one range of addresses is physically copied to a second range of addresses. Briton teaches about addresses[41] across the processor interface buss. Briton's addresses are not the network device addresses of the OSI Model, but are instead pure memory addresses.

Failure to use the 7 layer model

[84]. Briton's applications must be designed for the memory bus address and data-word lengths provided with a processor. Britton's FPGA (104) applications are then tightly coupled to the physical platform by the processor type on the memory bus. Britton provides no isolation of the memory bus physical features from the FPGA (104) applications; for example when processors moved to 16 bit buss widths from 8 bit buss widths, every one of Britons FPGA (104) applications needed to be redesigned for the new width. This would have happened again when processors moved to 32 bit buss widths. When Double Data Rate (DDR) memory came out, the faster signaling speeds and 64-bit data widths required advanced data-error functions to be included (every 8 data bits needed an extra 9th bit for the error

[39] Col 5, line 21.
[40] Col 4, line 45; Col 5, line 55.
[41] Col 4, lines 18 – 44 discuss the memory mapped addressing features.

recovery). Briton does not teach where the extra error coding functions go. Briton's FPGA (104) applications would have needed to be redesigned for the new width and the advanced data error functions. These historical changes to the memory bus (*i.e.,* through DDR) illustrate the failure of Briton's platform dependent approach vs. the open architecture approach and the platform independence that comes with the OSI seven layer model.

Operating System Dependencies Are Not Taught

[85]. There are critical operating system dependencies for Briton's data types that are missing; such as the configuration data for Briton's Micrprocessor Interface (100), such as a Part ID map that would be associated with the Part ID data of his System Registers (306) for software on the processor(102) to confirm the correct user space was loaded onto FPGA (104), and the download file that is needed to configure the user space of the FPGA (104), which will eventually set the Part ID data in Briton's System Registers (306).

Conclusion: Irreconcilable Incompatibilities between the Dretzka and Briton references

[86]. The Briton reference is very narrowly focused on non-standard custom interfaces between one processor (102) and an FPGA (104)[42]. By teaching that the application logic must interface with his address registers and data registers (306), Briton does not comply with the OSI seven layer model. By specifying that the FPGA (104) must interface directly with the processor (102)[43], Briton excludes applying his approach to the open systems interfaces of the OSI Model (e.g., Ethernet, PCI Express, or Rapid IO) and Briton excludes switched interfaces taught by Dretzka. The addresses of Briton's teachings are for registers (306) associated to memory addresses; Briton's addresses are not for devices associated to a network as Dretzka teaches. The equivalent OSI layers that Briton discusses only have a transmit stack in the processor (102) and only have a receive stack in the micro-processor interface (100) inside the FPGA (104); whereas Dretzka teaches that all units (10, 20, 30) on the network have both a transmit stack and a receive stack.

Contrast with the Dretzka Reference

[87]. The Briton and Dretzka references to interface types stand in technical opposition to each other, as summarized in the table below:

	Brition's interface	**Dretzka's interface**

[42] Col 1, lines 28 – 37, Col 2 lines 11 - 12.
[43] Col 1, lines 28 – 37, Col 2 lines 11 - 12.

1.	Internal – attaches directly to the processor (102) ♦ Across an internal memory buss	External – attaches indirectly to the processor (11, 21) through adapters (14-0, 14-4,14-5, 14-m, 24-0, 24-4,24-5, 24-n) ♦ Across a facility wide network
2.	Parallel Type with many separate circuits' lines for the address, data and clock signals. ♦ Only very short distances	Multiple Serial Bus Interface, where address, data and clock are transmitted over a singular circuit. ♦ Any distance
3.	Equivalent to Layer 1 and a fragment of Layer 2 Receive of OSI Model with only one instance per interface.	Specified as functions in Layers 2, 3 and 4 of OSI Model, with Multiple instances of Layers 1,2,and 3.
4.	Singular interface from 1 processor to 1 FPGA	A multiply-noded network, with multiple interfaces between each of the processing nodes
5.	One Processor to One or more FPGA	Many Processors to Many Processors
6.	Platform Dependent Applications	Platform Independent Applications

[88]. A designer would need to choose one but not both of the approaches taught by Briton and Dretzka.

1. By using Dretzka's external interfaces through an adapter, the processor is relieved of being overloaded by low level circuit interrupts from the network. The circuitry is much simpler using Briton's internal interfaces without an adapter, however the processor will have severely reduced resources remaining available for processing the application. These two approaches are incompatible; a designer would need to choose one approach over the other.
2. Dretzka's Serial Bus Interface is incompatible with Briton's direct parallel buss interface. A designer would need to choose one type of buss over the other.
3. Dretzka's multiple physical interfaces become cost prohibitive for parallel memory bus implementations like Briton's. The cost for the processor with multiple parallel memory interfaces will be extremely high, and would be very difficult to justify. In fact, industry has moved away from the multiple parallel memory bus processors, in favor of multiple serial memory bus processors. Processor designers have already chosen the serial type of buss over the parallel buss approach. It does not make sense to use the equivalent layer 1 from Briton with layers 2, 3 and 4 from Dretzka. If a designer is using Dretzka, they have chosen not to use Briton's equivalent of layer 1.
4. With Briton, there is only one pathway and one place for the FPGA to send data because there is only one processor and one buss. With Dretzka there is greater complexity because there are multiple pathways for

the data, and multiple destinations. These two approaches are incompatible; a designer would need to choose simpler approach over the other more complex approach only if the simplifying constraints can be met.

5. With Dretzka, there are only a multiple pathways to all the nodes. Every node has symmetric autonomy. Briton can not support this.

6. Briton's User Logic is highly dependent on the kind of memory interface provided by the processor, so Briton applications are platform dependent, becoming quickly obsolete. Dretzka fits into the OSI model that isolates the application logic from changes in the platform and networks, so application logic is platform independent and out survives the obsolescent hardware. These two different approaches are incompatible.

[89]. The addresses of Briton's teachings are registers or memory addresses, not the network addresses used by Dretzka or the OSI Model. The processor (102) of Briton's device must control all aspects of data transfer to and from the FPGA. Briton's applications in the FPGA user space depend upon the system registers (306) that are controlled by the processor (102). If we remove the processor (102) and somehow connect Dretzka's network lines in their place, there is nothing left in control of Briton's system registers (306) and Briton's applications are left out of control and without a way to send out a transaction to any network address. Mixing the two references in this way (Briton's reference to an equivalent OSI layer 1 with the Dretzka's reference for OSI layers 2, 3 and 4) will not work because they are incompatible.

[90]. Another alternative exists for mixing these two references: using the Briton reference to implement the upper layers of the stack, *i.e.*, layer 7, (Application - Network process to application), layer 6, (Presentation - Data representation, encryption and decryption) and layer 5 (Session - Interhost communication). However, Briton does not teach about implementing any of these functions and interfaces. Again, the processor (102) of Briton's device must control all aspects of data transfer to and from the FPGA (104). The "application" of Briton's teaching is required to utilize his physical address and physical data registers (306). The service element functions of the upper layers don't exist in the FPGA (104). Briton is silent on this issue, but if any of service element functions exist, they must execute in the processor (102). Briton's FPGA (104) application is by definition incompatible with OSI Model Applications in Layer 7. It is not possible to mix Briton's and Dretzka's teachings into an OSI model compliant FPGA implementation.

[91]. There is a second way to look at this alternative (using the Briton reference to implement the upper layers of the stack): compare Briton's teachings with the subset of functions integral to a Layer 5 Service-Element, which is the Layer above the Dretzka Reference. The Layer 5 functions list is repeated below:[44]

[44] ISO/IEC 7498-1: 1994(E) section 5.8.6 which starts on page 20.

Functions integral to a Layer 5 Service-Element	**Briton's Microprocessor Interface (100)**	**Unspecified Software – Processor (102) Might Run**	**Networking Protocol - Not Applicable (N/A) to Briton**
Connection Establishment & Release	Silent	N/A – bus is always connected	√
Suspend	Silent	A Possibility	√
Resume	Silent	A Possibility	√
Muxing & Splitting	Silent	A Possibility	√
Normal Data Transfer during Establishment	Silent on Layer 5 behaviors – only physical layer is taught[45]	A Possibility	√
Flow Control	Silent on Layer 5 behaviors – only physical layer is taught[46]	A Possibility	√
Expedited Segmenting	Silent	N/A	√
Blocking	Silent	A Possibility	√
Concatenation	Silent	A Possibility	√
Sequencing Acknowledgement	Silent	N/A	√
Error Detect & Notification	Silent	N/A	√
Reset	Silent on Layer 5 behaviors – only physical layer is taught[47]	A Possibility	√
Routing	Silent	N/A	√
Quality of Service	Silent	N/A	√

[92]. Using Briton's teaching, the only possible way to achieve some subset of Layer 5 Service-Element behavior is to write unspecified software to run inside the processor (102) to provide the behavior outside of the FPGA (104). Briton's FPGA (104) does not include a minimal set of service element behaviors for both stacks, and the application software can not communicate through the stacks to the application hardware.

---

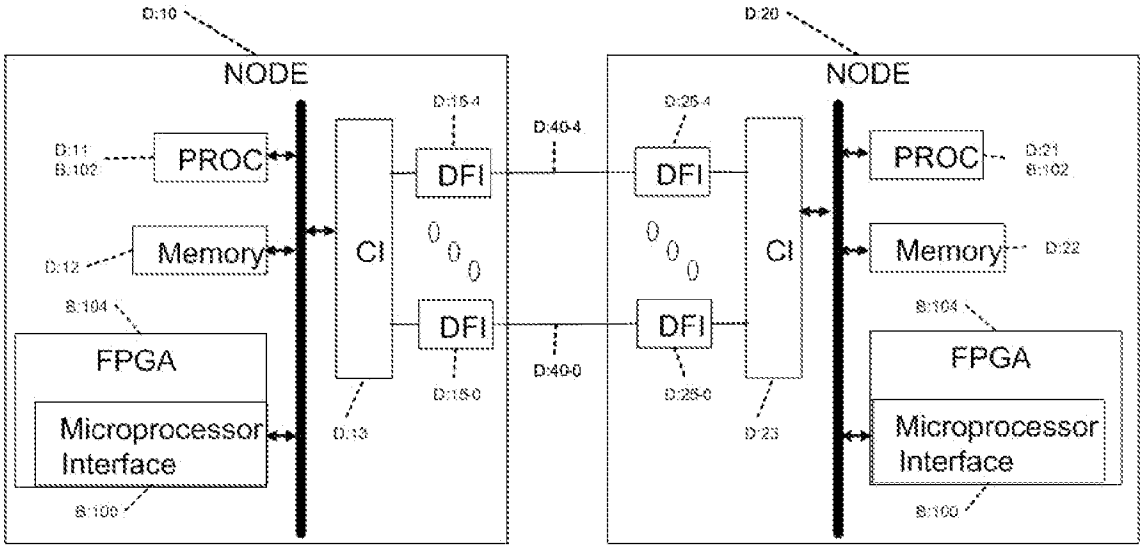
[45] Normal Data Transfer over a network will not typically include Memory Addresses with the Data, the Chip Select Signal, or the Read/Write Signal.

[46] Flow Control over a network will not typically include Memory Addresses with the Chip Select Signal, or the Read/Write Signal.

[47] Resetting a network device is significantly different behavior than resetting a register or a simple peripheral.

[93]. Another major obstacle (for one ordinarily skilled in the art) to join these two references is the non-obvious relationship between these two dissimilar transactions; a single network device address and data payload from Dretzka needs to transform into the continuum of memory addresses associated with a continuum of data from Briton.

[94]. There is another configuration of Briton and Dretzka which is derived by combining Briton's Figure 1 with Dretzka's figure 2; this is shown below.



[95]. In this last arrangement, the FPGA (B:104) is still slaved to the Processor (D:11/B:102 and D:21/B:102). Briton's Microprocessor interface (B:100) does not transfer data over the network (D:40-0 through D:40-4) using the CI (D:13) and DFI(D:15-0 through D:15-4). Neither Dretzka or Briton have teachings that explain how to put all of the service elements of a transmit stack and receive stack into the FPGA (B:100) to do transfer data over the network (D:40-0 through D:40-4). Again, Briton's FPGA (104) does not include a minimal set of service element behaviors for both stacks, and the application software can not communicate through these stacks to the application hardware.

[96]. I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 USC §1001 and that such willful false statements may jeopardize the validity of the above identified application and/or any patent issued thereon.

/ John W. Rapp /	May 28, 2010
John W. Rapp Residence: Manassas, Virginia Citizenship: USA	Date